Proceedings of GREAT Day

Volume 2010 Article 2

2011

Modeling With Differential Equations

Malcolm Kotok
SUNY Geneseo

Modeling With Differential Equations

Submitted by Malcolm Kotok

Abstract.

There are many useful mathematical tools that use differential equations to model harmonic motion and spring motion in particular. Our goal is to create a new, unique spring model tool that covers a broad variety of spring problems, is easy to understand with a coherent layout, and provides a degree of variability in the external forcing term beyond what existing programs provide. We compared this new tool and several existing ones. Many programs do not allow for external forcing terms, and those that do have put many restrictions on them. We identified mechanisms for entering general external forcing terms and solving the resulting differential equations, and incorporated those mechanisms into our program. The variability of the external forcing term allows this on-line program to solve an unlimited number of different exercises. Finally we looked at various future enhancements that could be implemented in the current program.

1. Introduction

The motivation for this project came from an intense love of both mathematics and computer science. We sought to help improve students' learning in the area of differential equations through the use of technology. We also wanted to create a new tool for teachers and professors to use when teaching about topics relating to differential equations and spring motion.

2. Differential Equations

In mathematics, the *derivative* of a function $dx/dt = f'(t)$ describes the rate of change of the original function f. A *differential equation* is an equation involving derivatives of an unknown function, and solving a differential equation means to determine what the unknown function is. Differential equations are used to model real world problems. Generally we are given two pieces of information: initial conditions and a description of how things change. Initial conditions describe the state of our system at a particular time, and information about the derivative describes how our system is changing. From this information we produce a differential equation through a process called modeling. Next we can solve the differential equation using three different techniques. Optimally, we solve it analytically, which means we are able to find an exact formula for the solution. If this is impossible or too difficult we can approximate the solution either through qualitative or numerical analysis. *Qualitative analysis* approximates the solution graphically and *numerical analysis* determines an approximate solution through iterative algorithms usually performed on a computer.

The particular model we are using in this paper is an example of an *ordinary differential equation*, which is a differential equation depending only on a single independent variable. In this case x will be our dependent variable and t will be our single independent variable. Using differential equations we can predict future behavior of the object or quantity being studied, in this case spring motion. For more information see [Edwards, Section 1.1].

3. The Spring Model

Harmonic motion is a specific real world situation that can be modeled with differential equations. One type of harmonic motion is concerned with the motion of a spring. The program we created solves this model of spring motion using numerical methods. First we must determine what differential equation describes this situation.

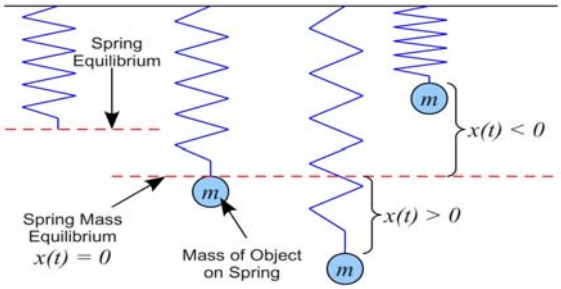

Figure 1: The Spring Model

As shown in Figure 1, we have a spring hung downward attached to an anchor device at the top. Without anything attached to the spring, it comes to rest at what is called *spring equilibrium*. When we attach an object of mass m to the bottom of the spring, it comes to a new, lower rest position called *spring-mass equilibrium*. We denote the displacement, or distance, the mass is from spring-mass equilibrium by $x(t)$. Stretching the spring beyond spring-mass equilibrium corresponds to a positive value of $x(t)$. Compressing the string corresponds to a negative value of $x(t)$. We want to model the motion of the object attached to the spring. To do this we must look at the forces involved that act on the spring.

One force acting on this system is the *spring force*, denoted by F_S. Hooke's Law says F_S is proportional to the displacement, and is opposite of displacement. That is

$$F_S = -kx$$

for some positive number k known as the spring constant. Essentially this determines how easily the string stretches and compresses. Another force involved is known as the *damping force*, denoted by F_D. Damping is due to friction that might occur in the system such as air resistance. One simple way to model damping is to assume that damping is proportional to the velocity and again acts in the opposite direction. This is

$$F_D = -dv = -dx'$$

for some non-negative number d known as the damping constant. (Note that velocity is the derivative of position. That is, $v = x'$.) If $d = 0$, then there is no friction involved in the model. Last, we consider a possible external force, denoted by F_E. This is a force from an external source and does not depend on the spring or the attached mass, and it is some function of t. For example, the spring system could be attached to a motor that oscillates the anchor device. That is,

$$F_E = b(t)$$

where $b(t)$ is a function that describes the external force. To put these together we must use Newton's second law of motion. Simply put it states that the sum of the forces acting on system must equal the mass times acceleration. That is

$$ma = mx'' = F_S + F_D + F_E$$

where m is the mass of the object. (Note that acceleration is the second derivative of position. That is, $a = x''$.) Substituting the equations from above we get

$$mx'' = -kx - dx' + b(t)$$

or

$$mx'' + dx' + kx = b(t)$$

This is the differential equation that models spring motion, where m is the mass of the object hung on the spring, d is the damping constant, k is the spring constant, $x(t)$ is the displacement at time t from spring-mass equilibrium, and $b(t)$ is some external force as discussed before.

When there is no external force, $b(t) = 0$, this differential equation can be solved analytically and an explicit formula for the solution can easily be determined. However, as soon as $b(t) \neq 0$, finding exact solutions can sometimes be difficult. To allow for the arbitrary external forcing terms, we must resort to numerical analysis to approximate a solution. The goal of this project is to create an on-line applet that can solve the spring model in general for any values of m, k, d, and $b(t)$ and then provide a animated graphical interpretation of the solution. Our most important goal was to accurately solve the spring equation when the external forcing term is non-zero. The resulting applet is pictured in Figure 2. For more information see [Edwards, Section 2.4].

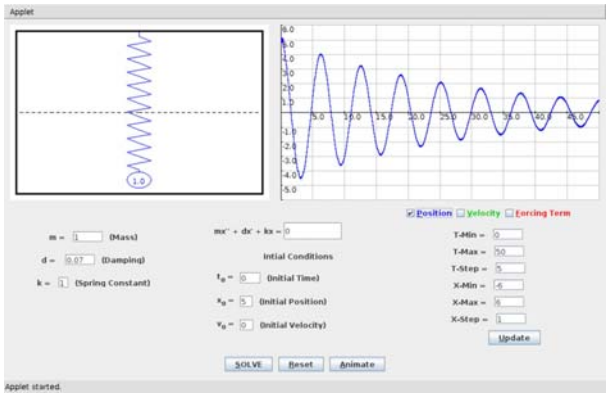

Figure 2: The Spring Motion Applet

4. The Spring Motion Applet

We achieved our goals of creating an applet that accurately approximates solutions to the spring model with arbitrary values of the forces discussed by writing a parser and interpreter for arbitrary forcing terms, and using the Runge-Kutta method to solve the resulting differential equations. Figure 2 shows the complete applet. It can be accessed on the internet through the link address of http://go.geneseo.edu/springmotion.

Notice the various functions of the applet. Initially, a sample equation is provided, with the solution graphed on the right-hand side of the window. Also provided is a drawing of the spring which can be animated. Different values of m, k, and d can be entered on the bottom left side of the applet, as well as different initial conditions in the middle of the bottom. The applet will accept any positive values for m and k, and any non-negative value for d. Also notice that the solution window-size can be changed on the bottom right side of the applet. This software is capable of graphing the position or velocity of the spring as well as the external forcing term acting on the system. With any of these, it is possible to animate the solution as time progresses. Figure 3 shows the applet mid-animation at approximately $t = 17$.

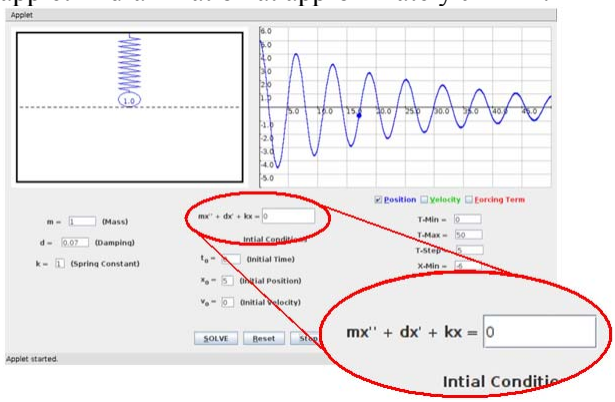

Figure 3: Mid-Animation of the Spring Applet and a Magnification

The most significant part of this program is the differential equation in the center which is magnified in Figure 3. This is the spring model with a box on the right-hand side of the equation where the user can input an external forcing term. This software is capable of accepting many external forcing terms and generating numerical solutions.

5. Comparison to Other Available Applets

We are unaware of any publicly available applets on-line that meet all of the objectives we set out to accomplish. We have classified the current programs into three categories: applets with no external forcing term, applets with one external forcing term, and applets with a small selection of external forcing terms.

5.1 Applets with No External Forcing Term

Some applets have the ability to solve the spring model for various values of m and k and then provide a graphical interpretation of the solution. They do not provide any means of solving differential equations with an external forcing term. There are many of these types of applets available on-line. In particular,

- "Spring Pendulum" (http://www.walter-fendt.de/ph14e/springpendulum.htm),
- "Damped Oscillator" (http://www.lon-capa.org/~mmp/applist/damped/d.htm), and
- "Spring Mass Applet" (http://www.ibiblio.org/links/applets/appindex/springmass.html)

are three examples of this kind of applet. Some of them (for example, "Damped Oscillator") allow for various values of the damping constant d as well. Most of the applets found on-line are of this type, and so there are many different variations. However, they cannot accept an external forcing term, and this limits their capability. These types of applets do not accomplish our goal.

5.2 Applets with One External Forcing Term

Some applets have the ability to solve the spring model for various values of m, k, and one particular external forcing term and then provide a graphical interpretation of the solution. One such model that does this is called "Forced Oscillations (Resonance)." (http://www.walter-fendt.de/ph14e/resonance.htm) This specific applet solves the spring model with a sinusoidal forcing term. Again there are restrictions to these types of applets. There are far fewer models available that are of this type, and they can solve only simplistic spring models. None that we found allow a change in damping, for example. Most importantly these applets cannot accept multiple external forcing terms. They allow only one. Therefore, these types of applets do not accomplish our goal.

5.3 Applets with a Small Selection of External Forcing Terms

Some applets have the ability to solve the spring model for various values of m, k, d and a select assortment of particular external forcing terms and then provide a graphical interpretation of the solution. As an example, "Differential Equation Applet" (http://www.falstad.com/diffeq/) is an applet of this type. This software allows external forces such as step functions, sine functions, exponential functions and linear functions. Figure 4 is a screenshot of this applet solving the differential equation with a sinusoidal forcing term.

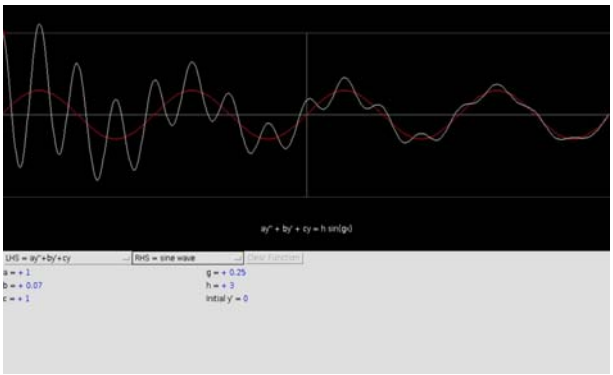

Figure 4: A Differential Equation Applet Example

Even fewer of this type of applet are available on-line. In fact, this was the only applet of this type we were able to find. Notice also that this particular applet does not provide an animation of the system, just a graph of the solution curve with the external force. Although these types of applets do provide a few choices for an external forcing term, they are not able to accept generic external forcing terms. These types of applets do not accomplish our goal, but they do come close.

6. Mechanisms for Providing General External Forcing Terms

So how is our program capable of solving differential equations with the most

generic external forcing terms? To understand this better we split the process into two components: numerical analysis and interpreting user input as mathematical functions.

6.1 Numerical Methods

Mathematically we need to solve the differential equation given by our spring model for the most general external functions. As stated before, in most cases we cannot use analytical methods to get an explicit formula for the solution. As a result, we must resort to estimating the solution with numerical analysis. To understand the numerical analysis in solving differential equations better we turn to the building block of it: *Euler's Method*.

Euler's Method is applied to a differential equation of the form $dy/dx = f(x,y)$, with an initial condition. Here is how the method works. We know one exact point on the solution curve, the initial condition. We call this point (x_0, y_0). From this point we go a small distance left or right to x_1. This distance is called the step size, denoted h, and the smaller it is the better the approximation. To calculate y_1, we use the fact that we know the derivative as part of the differential equation. This derivative tells us the slope of the tangent line, and we can follow that to get our approximation. From this new point (x_1, y_1) we repeat the process. In Figure 5, the gray curve represents the solution and the black points our estimate.

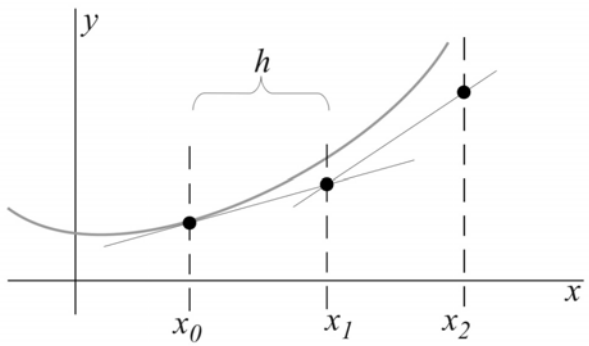

Figure 5: Euler's Method

If we are given the initial value problem of $dy/dx = f(x,y)$ with $y(x_0) = y_0$, then Euler's Method can be produced with the iterative formulas given by

$$x_{n+1} = x_n + h$$

$$y_{n+1} = y_n + h \cdot f(x_n, y_n)$$

As one might notice in Figure 5, using this method leaves some error between our approximation and the actual solution. To fix this we improve our algorithm. We will now look at the *Improved Euler's Method*, sometimes called the *second-order Runge-Kutta method*. This method works similarly. We still start at our initial condition (x_0, y_0), we still compute the slopes using the derivative, and we still have some defined step size h. However, each iteration of this method requires three steps. The first two of these steps is essentially a single iteration of Euler's Method. First we use Euler's method to find a point (x_1, u_1). From this new point we use Euler's Method to get another point (x_2, u_2). These would correspond to (x_1, y_1) and (x_2, y_2) in Euler's Method. However, once we have our second point, we determine a slope between this point and our original point and determine our y_1 by this new line at x_1. In Figure 6, the gray curve again represents the solution, the gray points our estimate and the black dotted line the slope used to get our second point.

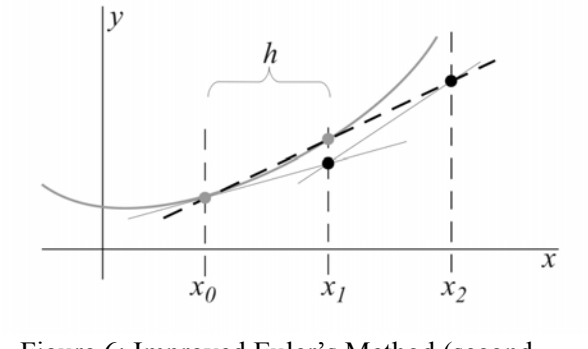

Figure 6: Improved Euler's Method (second-order Runge-Kutta)

Now, our approximation is much better for the same step size. This is classified as

second-order since, we need to calculate the average of two slopes. Our program uses the *fourth-order Runge-Kutta method*, which calculates 4 slopes and averages them.

Mathematically, we almost have a way to approximate the solutions to these generic differential equations. However one more problem remains. The methods described above only works on first-order differential equations. This means the differential equation only relates the function with its first derivative. Second derivatives and higher-order derivatives do not appear in the equation. Our spring model

$$mx'' + dx' + kx = b(t)$$

is a second-order differential equation. To fix this problem we can use substitution to create a system of two first-order differential equations. These two first-order differential equations will relate to one another in a way that produces the original second-order differential equation. With these two new first- order differential equations, we can now apply our methods to each and determine an approximate solution to our spring model for any external force $b(t)$.

For example, if we introduce two new variables, u and v, with $u = x$ and $v = x'$, we can get a system of first-order differential equations in terms of u and v. We see that $u' = x' = v$, and $v' = x''$. If we substitute our spring equation we get

$$v' = x'' = \frac{-dx' - kx + b(t)}{m}$$

$$= \frac{-k}{m}u + \frac{-d}{m}v + \frac{b(t)}{m}$$

Thus we have written our spring model as a system of two first-order differential equations. If we write this as a matrix-vector equation we have

$$\begin{pmatrix} u \\ v \end{pmatrix}' = \begin{pmatrix} 0 & 1 \\ -k/m & -d/m \end{pmatrix} \cdot \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} 0 \\ b(t)/m \end{pmatrix}$$

From here, we apply our numerical methods on both first-order equations simultaneously. For more information see [Edwards, Chapter 6].

6.2 Parsing User Input

Now that we understand how to approximate solutions to these general differential equations, we need to understand how the program interprets user input. The second requirement of the project was to create a way to accept text from the user as mathematical functions. The program needed to interpret user defined functions and input different values for variables (t and x in our case). This requires a three stage process as shown in Figure 7. First we must break up the input text into "words". Then we must parse it into something the computer can understand. Last we must store the information. The text is analyzed through a process called *lexical analysis*. The text is parsed by what is known as an *operator-precedence parser* and stored into what is called a *syntax tree*.

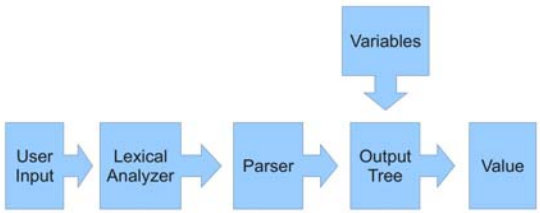

Figure 7: User Input Flow

Input is sent to the lexical analyzer first. The lexical analyzer splits the text into parts that the parser will recognize. These pieces are known as *tokens* or *lexemes*. The analyzer needs to recognize a string of characters such as "`cos`" as a single token and not "`c`", "`o`", and "`s`" all separately. Similarly it needs to recognize an input string of "`ttanx`" as "`t`", "`tan`", and "`x`", and not all as one word. To accomplish this, the analyzer must first have a stored list of what strings of characters are acceptable. From here, the analyzer reads one character at a time and eliminates any string that does not start with that character. It then looks at the second character and repeats the process. If it comes to a character that only matches one string in its current list, it assumes that to be the token, and finishes checking the input characters

to be certain. If it comes to a character that does not match any string in its current list then there are two options. Either, the input is an error, or the token was eliminated in the previous list. The analyzer checks both these possibilities. Once this is done, the analyzer passes a list of these tokens to the parser.

The operator-precedence parser now needs to interpret these tokens so they make sense mathematically. From the tokens the parser builds a tree structure that unambiguously captures the syntactic structure of the original expression, following the order of operations we know from basic arithmetic. For example, if the parser receives input from the analyzer as `{3 + 4 * 5}`, it needs to create a tree that multiplies 4 and 5 first, and then adds 3. The parser goes through the input tokens one by one and produces an output tree that can be used elsewhere.

The parser can accomplish these goals through what is known as a *grammar*. With this grammar, one can generate a table telling the parser how to compare the tokens it is receiving from the lexical analyzer. For example, our table tells the parser to evaluate expressions inside parenthesis before proceeding to read more tokens. Likewise, it tells the parser to evaluate numbers before trying to add or multiply them.

The output tree is the final step in our process. The tree is traversed from top to bottom, formally known as *post-order traversal*, and this algorithm will produce a value for the tree. In Figure 8 for example, the algorithm considers the input expression as simply the multiplication of two things. As we move down the tree we see the left branch, called a *node*, is a number, 3, and the right is a function of another node, in this case the variable t. The trees have the capability to hold variables as well. Whenever the algorithm comes across one of these nodes it will substitute the given value for that variable such as π. Now we can enter any expression, enter values for various variables, and get back the value of the expression. This completes our ability to read in text and evaluate it as an expression. For more information see [Parsons, Chapter 2] on lexical analysis, [Parsons, Chapter 3] on grammars, and [Parsons, Chapter 4] on operator-precedence parsing.

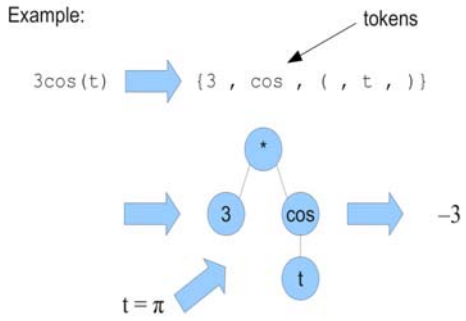

Figure 8: User Input Example

7. Conclusion

There are currently many on-line tools for solving differential equations arising in particular models, specifically spring motion. However, many of these tools are limited in the number of equations they can solve. Our goal was to create a new on-line tool that models spring motion for general equations and then provides a graphical interpretation and animation of the system. Most importantly, this tool should allow for generic external forcing terms.

In the future different applications beyond springs could be added, such as pendulums, animation speed could be adjusted by the user, and the program could be tested with actual users to improve it even further. For now, a useful tool exists that students and teachers around the world may use to better their understanding or teaching of differential equations and spring motion.

References

Edwards, Henry C., David E. Penney, and David Calvis. *Elementary Differential Equations*. 6th ed. Upper Saddle River (New Jersey): Prentice Hall, 2008. Print.

Parsons, Thomas W. *Introduction to Compiler Construction*. New York: Computer Science, 1992. Print.